Code Hunt: Experience with Coding Contests at Scale

Judith Bishop
Microsoft Research
Redmond, WA, USA
jbishop@microsoft.com

R. Nigel Horspool
University of Victoria
Victoria, BC, Canada
nigelh@cs.uvic.ca

Tao Xie
University of Illinois at Urbana-Champaign
IL, USA
taoxie@illinois.edu

Nikolai Tillmann,
Jonathan de Halleux
Microsoft Research
Redmond, WA, USA
nikolait,
jhalleux@microsoft.com

***Abstract*—Mastering a complex skill like programming takes many hours. In order to encourage students to put in these hours, we built Code Hunt, a game that enables players to program against the computer with clues provided as unit tests. The game has become very popular and we are now running worldwide contests where students have a fixed amount of time to solve a set of puzzles. This paper describes Code Hunt and the contest experience it offers. We then show some early results that demonstrate how Code Hunt can accurately discriminate between good and bad coders. The challenges of creating and selecting puzzles for contests are covered. We end up with a short description of our course experience, and some figures that show that Code Hunt is enjoyed by women and men alike.**

***Index Terms*—Programming contests, unit tests, symbolic execution, Code Hunt game**

I. *Introduction*

Two of the backbones of software engineering are programming and testing. Both of these require many hours of practice to acquire mastery. To encourage students to put in these hours of practice, educators often employ the element of fun. Generally, this involves setting engaging assignments which emphasize the visual, audio, mobile and social world in which the students now live. However, a common complaint in second or third year is that "students can't program" which is usually interpreted as meaning they are not able to produce code readily for fundamental algorithms such as read a file or search a list. Recruiters in industry are famous for requiring applicants to write such code on the spot. Thus there is a dichotomy: how to maintain the self-motivation of students to practice coding skills, and at the same time focus on core algorithmic problems.

An answer is to use the challenge of a game. Games are everywhere these days, and the motivation to score, do better and complete the game is very high. We are familiar with the concept of playing against the computer, and the sense of achievement that is acquired when goals are reached or one wins. Winning is fun, and fun is seen as a vital ingredient in accelerating learning and retaining interest in what might be a long and sometimes boring journey towards obtaining a necessary skill.

In the context of coding, there have been attempts to introduce fun by means of storytelling [8], animation (www.scratch.mit.edu) and robots (e.g. www.play-i.com). Code Hunt adds another dimension – that of puzzles. It is with these ideas in mind that we conceived Code Hunt, a game for coding against the computer by solving a sequence of puzzles of increasing complexity. Code Hunt is unique among coding systems and among games in that it combines the elements of both to produce just what we need to get students to put in those hours of practice to hone their programming skills. Along the way, they also learn to understand testing, since the game is based on unit tests. Code Hunt has been used by over 50,000 players, and we have figures to support the claim that they enjoy the game, stay with it, and acquire mastery in coding.

Learning to code by solving puzzles is not the same as learning to code by writing to a specification. There are many contests where students pit their wits against each other – and against the clock – to create a solution to defined problems. This kind of coding is similar to that which they encounter in courses or later in their careers. Code Hunt is different in that learning to code is a by-product of solving a problem which is presented as pattern matching inputs and outputs. The fun is in finding the pattern.

In previous work, we discussed the technical challenges of building Code Hunt, as well as its predecessor Pex4Fun [12]. [13] [14] [14]. This paper concentrates on the insights that we are acquiring into the behavior of players through controlled play via contests. We have run 14 contests to date, with varying sizes and audiences, and are beginning to gain some understanding of the motivating and demotivating factors that can affect performance in the game. Our figures feed back into further contests and improve the experience for players, as happens with most games.

In Sections II and III we describe Code Hunt, the game, as well as its architecture built on Azure. Sections IV and V present our results of running contests on Code Hunt, and how to create a contest. Section VI discusses briefly how Code Hunt can also be used for courses. Sections VII and VIII wrap up with Related Work and Conclusions.

II. *Background to Code Hunt*

Code evaluator systems are very popular, with the growth in student numbers and the popularity of MOOCs. These systems work on the basis of a problem specification and a set of test cases to establish if the student has achieved an acceptable

program. Several years ago, we released Pex4Fun www.pex4fun.com which did the opposite: presenting an empty slate to the user and only a set of constantly changing test cases [13] – there is no specification. To solve a puzzle (called a duel) in Pex4Fun, the player iteratively modifies code to match the functional behavior of a secret solution. The player is guided by the set of test cases automatically generated by a white-box testing tool called Pex [11]. These show for a selection of sample inputs when the player's code and secret code have the same outputs or different outputs. As a state-of-the-art implementation of dynamic symbolic execution, Pex [5] conducts path exploration partly guided by fitness values computed via a fitness function.

Although Pex4Fun was, and is, very popular, we wanted to extend its capabilities as a game and investigate how far we could retrofit the data that is mined to provide hints to the player. We also wanted to bring the game to a larger audience with more languages. Thus Code Hunt was born (Figure 1).

Figure 1 The opening screen of Code Hunt

III. *Code Hunt*

A. Overview

Code Hunt is a serious game where the player has to write code to advance. Code Hunt runs in any modern browser at www.codehunt.com; see Figure 1 for the splash screen. The built-in tutorial reveals the following story to the player:

> Greetings, program! You are an experimental application known as a CODE HUNTER. You, along with other code hunters, have been sent into a top-secret computer system to find, restore, and capture as many code fragments as possible. Your progress, along with your fellow code hunters, will be tracked. Good luck.

The game is structured into a series of sectors, which in turn contain a series of levels. In each level, the player must write code that implements a particular formula or algorithm.

As the code develops, the game engine gives custom progress feedback to the player. It is part of the gameplay that the player learns more about the nature of the goal algorithm from the progress feedback. Figure 2 shows the feedback loop between the player's code in the browser and cloud-based game engine.

The player can write code in an editor window, using either C# or Java as the programming language. This code must implement a top-level function called "Puzzle". The puzzle has some input parameters, and it returns a result. The player has only one way to test if the current code implements the goal algorithm: by pressing on a big "CAPTURE CODE" button.

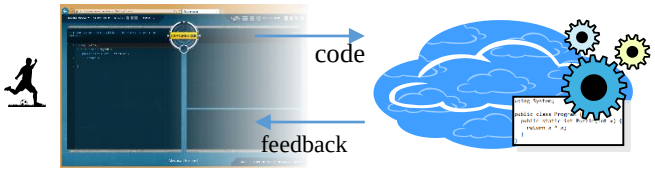

Figure 2 Game play

Pressing this button causes a chain of events:

1. The code is sent to a server in the cloud.
2. The server compiles the code (including a Java-to-C# conversion when required).
3. The server starts an in-depth analysis of the code, comparing it to the goal algorithm.
4. The results are returned and shown to the player.

The result is either a compilation error, or a list of mismatches and agreements with the goal algorithm. Figure 3 shows the code on the left, and the mismatches (red crosses) and agreements (yellow checkmarks) are shown on the right.

Figure 3 The Code Hunt main page, showing test results

If the code compiles and there are no mismatches and only agreements with the goal algorithm, the player wins this level – or as the game puts it, the player "CAPTURED!" the code, as shown in Figure 4.

Figure 4 After solving a puzzle, the player gets a score

The in-depth analysis returns each mismatch and agreement with the goal algorithm in the form of a tuple (input, actual result, expected result).While the actual and expected result are the same when the player's code is in agreement with the goal algorithm, they are different when there is a mismatch. The player must inspect the mismatches and determine how to change the code to make it more like the goal algorithm. In other

words, the player must reverse-engineer a secret algorithm, and write semantically equivalent code.

B. Skill ratings and score

When the player successfully completes a level, the Code Hunt game engine assigns a "skill rating" to the player's code. The rating, an integer 1, 2, or 3, reflects the elegance of the solution, measured by its succinctness (a count of instructions in the compiled .NET intermediate language). 1 indicates that the solution is much longer than most other submitted solutions, 2 means about average, and 3 means significantly shorter.

The intention behind the skill rating is that it may motivate players to keep tinkering in a particular level in order to improve their code, thus greatly extending the gameplay time. This rating is multiplied by a level-specific value that reflects the difficulty of the level, resulting in the "score" for this level.

Figure 4 shows the rating 1, and a score of 2 (implying a multiplier of 2 for this level), after the player completed a level. Players can track their progress via an accumulated total score. To determine whether the player's code implements the goal algorithm correctly, Code Hunt leverages the white-box testing of Pex, which implements an automated program analysis technique based on dynamic symbolic execution.

C. Architecture

Code Hunt is a true cloud-based system hosted in Windows Azure. A player requests the page www.codehunt.com which is served from a front-end cloud app. If the player chooses to log in, Windows Azure Active Directory Access Control delegates authorization to one of the identity providers (Microsoft, Facebook, Google, Yahoo). Once the player engages in any particular level, a back-end cloud app is invoked at api.codehunt.com. The back-end exposes a publicly accessible REST-based service API which performs the actual program analysis tasks, and also persists user-specific data in Windows Azure Store. Guarded by an OAuth v2 authorization scheme, the back-end is available for use by other clients. We welcome other researchers who are interested in using this platform for other research topics.

Both the front-end and the back-end have been designed for maximum scalability, dynamically increasing the number of cores available to serve an arbitrary number of users. To illustrate the need for scalability, consider that each concurrent user of Code Hunt who presses the "CAPTURE CODE" button as part of the gameplay potentially causes a single core of the back-end to be busy analyzing the submitted code for up to 30 seconds. Many cores are necessary to support the users at peak times (during a contest), while very few cores may be needed at other times.

IV. *CODE HUNT FOR CONTESTS*

Soon after Code Hunt was launched in March 2014, there was interest from three different groups in Microsoft to use the tool in a contest environment. Put together, the accumulated advantages of Code Hunt were seen as:

1. The browser-based nature of the game gave it a world-wide reach on all platforms.
2. Automatic grading of the contest is cost effective.
3. Cloud based hosting means the ability to scale to thousands of players or more.
4. Prestige of the tool coming from Microsoft Research gives trust in its accuracy.
5. Clear scoring criteria implies preciseness of determining winners.
6. Results after the contest can be used by recruiters to get back to top coders.
7. The contest is fun, fresh and different.

We therefore embarked on partnerships to run online contests world-wide. We collected data on how students performed and the effect of puzzle difficulty on their performance. Our early results and observations are summarized in this section.

A. Puzzles and their difficulty

Puzzles are what Code Hunt is all about. We have a Puzzle Bank which is continually refreshed, since a contest requires puzzles that students have not seen before. For a contest, we need between 6 and 24 puzzles, depending on the length, in hours, of the event. When storing a puzzle in the bank, we annotate it with various properties, i.e.:

1. ID number: an integer
2. Group: numbers, strings, bools, arrays, binary
3. Description: a sentence describing the puzzle
4. Source: initials of the puzzle designer
5. Difficulty: a subjective rating between 1 and 5

A typical puzzle might be:

P067 arrays Remove duplicates from an array APCS 2

For the integrity of the game and contests, the descriptions are highly protected, and are known only to the puzzle and contest designers.

Once the time period of a contest is decided, the contest designers set about creating, adapting or re-using puzzles. In this process, the difficulty values are critical. For a successful contest, one needs to have easier puzzles in the early sectors, leading up to more challenging puzzles in latter sectors. The average difficulty of all puzzles in a typical contest is usually around 2.5, but skewed as described.

The challenge is that contests need to have mostly new puzzles, and the difficulty that we assign to them is subjective.

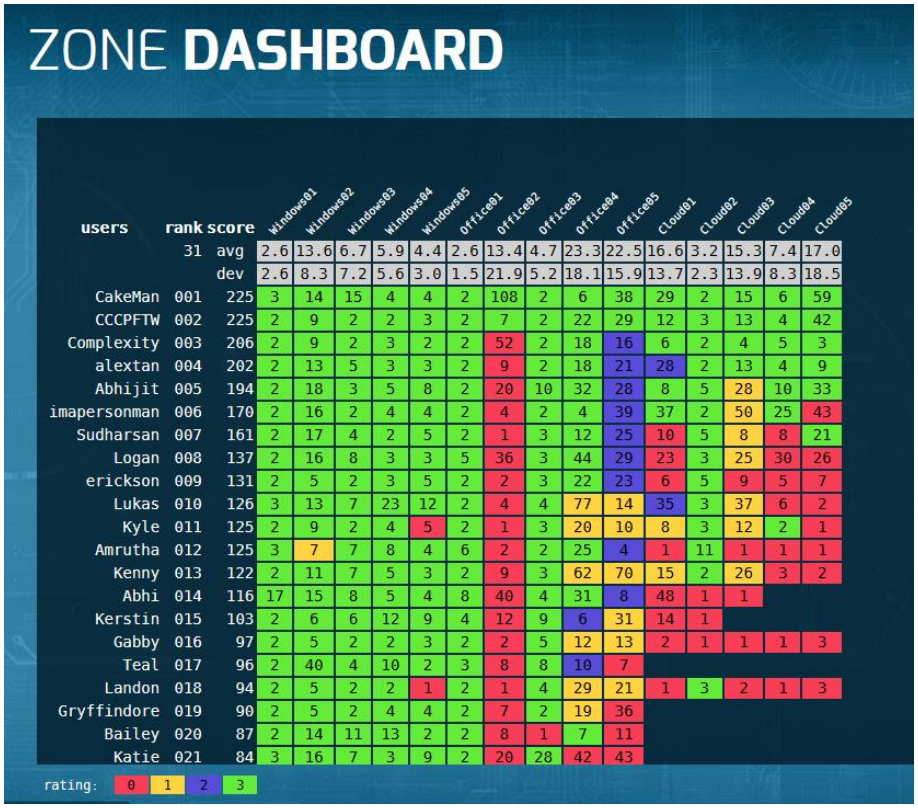

Figure 5 Dashboard from a contest

We do not know in advance how they will be perceived by players. Early on, we discovered that there were puzzles that simply fooled most players. They got stuck on that level and could not continue, or gave up. This effect can be seen in the dashboard from a contest shown in Figure 5. (The red squares indicate puzzles which have been attempted but not yet solved.) Clearly, some feedback into adjusting the difficulty factor would help in determining a puzzle's future use.

The score that players receive for solving a puzzle is, as described above, based on a rating of 1, 2 or 3. That was deemed too rough a measure for adding information to puzzles. The other metric we record is the number of attempts it took a player to solve a puzzle. Solving a puzzle in Code Hunt involves two steps: first recognize the pattern represented by the clues, and then code the algorithm. Pressing the "Capture Code" button is used by players to get more clues so that they can discover the pattern. The column of red ratings in Figure 5 indicates that most players had not yet solved the puzzle at all. Two players had solved the puzzle (green rating) but one of them took 108 attempts to get there. While extreme, it is not unusual for players to spend this long on a puzzle.

B. Tries as a difficulty measure

An alternative to average score achieved for a puzzle is therefore to look at the average tries made. In an initial validation of this hypothesis we examined a large contest and found the results as in Table I. Thus the top players (as determined by score) spent five or more times fewer tries on solving a puzzle than others. This bias has been maintained in further contests, though not always as dramatically. The ratio depends on several factors that we have ascertained from speaking to students and from surveys:

1. mix of students who enter
2. the internet speed
3. experience with using a powerful IDE

TABLE I AVERAGE TRIES ACROSS LAYERS AND TOP PLAYERS

Group	Count	Average tries per level
All players	2,353	41.0
Top players	350	7.6

In the case of 3, students are known to develop code offline and submit it to Code Hunt when the compilation is error free, for example. For 2, with a slow connection, students spend more time studying the clues and their code, and press the Capture Code button less often.

However, the scale of our data is considerable, and is able to smooth out these effects, we believe.

C. Adjusting puzzle difficulty

Thus we use average attempts (or tries) for a puzzle in a contest to calculate a new difficulty for the puzzle. Using a weighted average over the number of players who solved the puzzle, we feed that information back into the puzzle bank.

Our formula for the new perceived difficulty is:

*a + tries/b + tries/c * distance*

where *a* = 1, *b* = 20, *c* = 50 and tries is the average attempts for all players who solved the puzzle and distance is the number of levels solved so far. The last factor in the equation gives a weighting to the skill of a player: they are assumed to get better at coding as they solve more puzzles.

Using this formula, we can recalculate the Perceived Difficulty, D, of any contest, over all users and puzzles. Two examples stand out. In Table II we show a contest run in April 2014 over four rounds. Except for the first qualifying round, the puzzles were perceived as easier by the large numbers of players.

TABLE II CONTEST A - SAME COMMUNITY

Contest A (same community)	Subjective difficulty	Perceived difficulty	Players who started
Qualification	1.59	2.72	13773
Preliminary A	2.17	1.84	1017
Preliminary B	2.50	1.84	141
Semi-Final	2.60	2.22	1164

In another example, Table III, we gave the same contest to two different groups and they perceived it somewhat differently, although both spent more attempts on the puzzles than we would have expected. Here the sample size is much smaller. In adjusting the difficulty of puzzles, we take the number of players into account.

TABLE III CONTEST B - DIFFERENT COMMUNITIES

CSTA and TEALS (identical contests)	Subjective difficulty)	Perceived difficulty	Players who started
Students	1.96	5.22	61
Teachers	1.96	4.38	14

D. Outliers

We would like to know whether there are certain kinds of puzzles that are consistently more difficult than others. Consider these examples:

Players	Average tries	Sector.level
1683	3.88	3.3
376	45.08	5.2

Computing the difficulty according to the formula above gives:

D = 1 + 3.88/20 + 3.88 * 14 / 50 = 2.68
Original difficulty estimate was 2

D = 1 + 45.08/20 + 45.08 * 25 / 50 = 25.79
Original difficulty estimate was 2

In both cases, the perceived difficulty, D, increased, but by markedly different amounts. With a perceived difficulty of 26 (rounded up), the second puzzle qualifies as an outlier. We therefore increase its difficulty in the bank, but we also examine it along with other outliers for common qualities.

In our bank of 250 puzzles, we have around 15 puzzles that qualify as outliers. From the contests we have run so far, we cannot make any firm conclusions about common factors, because most of the puzzles are not used more than once, and therefore we do not have corroboration as to their perceived difficulty across player populations. However, applying this process in another part of Code Hunt has revealed more definitive results.

E. The effective of difficulty on player drop off rates

In addition to contests, which are put up for a set time of up to 48 hours, Code Hunt also has a default zone, where anyone can play and learn programming. The puzzles in the default zone follow the APCS computing curriculum [1] and start with arithmetic, loops, strings, arrays and so on. The figures for this zone are impressive: over 45,000 have started playing, and 120 players have completed all 130 puzzles. There is a drop off rate which is about constant at 15% initially then decreases as players become more expert and want to finish the game. Yet there are some dramatic drops and we examined what might be causing these.

Figure 6 shows the drop off rate for just the first 3 sectors. In Figure 7Figure 7 Puzzles with drop off rate higher than 15%, we can detect three effects where the previous puzzle caused more than a 15% drop off. These are itemized in Table IV Puzzles that cause a high drop off rate

TABLE IV PUZZLES THAT CAUSE A HIGH DROP OFF RATE

Color	Puzzles	Description
Yellow	Previous: 1.5, 1.6, 1.10, 1.15	Division
Blue	Previous: 1.12, 3.1	Unusual operators
Green	Actual: 1.1, 21.1, 2.2 Previous: 3.4	Start of sector and start of loops Unusually tricky puzzle

These results, based on tens of thousands of players (and hundreds of thousands of programs) taken over a six month period are surprising, but definitive. It would seem that division (which did not occur in other puzzles in these sectors) is difficult for players to detect. Our reading is that they just do not think of it, but we would need to do a more detailed study to confirm this hypothesis. Unusual operators such as mod and the bitwise binary operators (namely ~, &, | and ^) are also problematic. Finally, there is a drop off effect in this zone when new concepts are introduced. Code Hunt is not a teaching game – it is for drill and practice and learning about algorithms and testing. At present, there is no teaching material on the site. Thus students who move from the Arithmetic sector to the Loops sector, might not be able to cope.

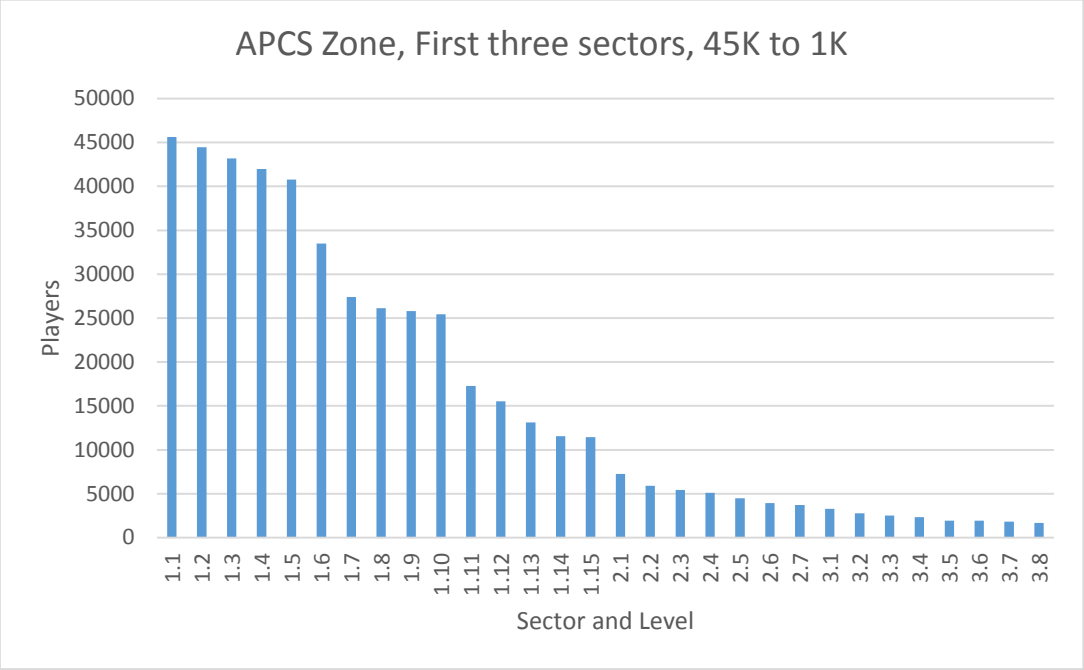

Figure 6 Drop off rate in the APCS Zone, first three sectors

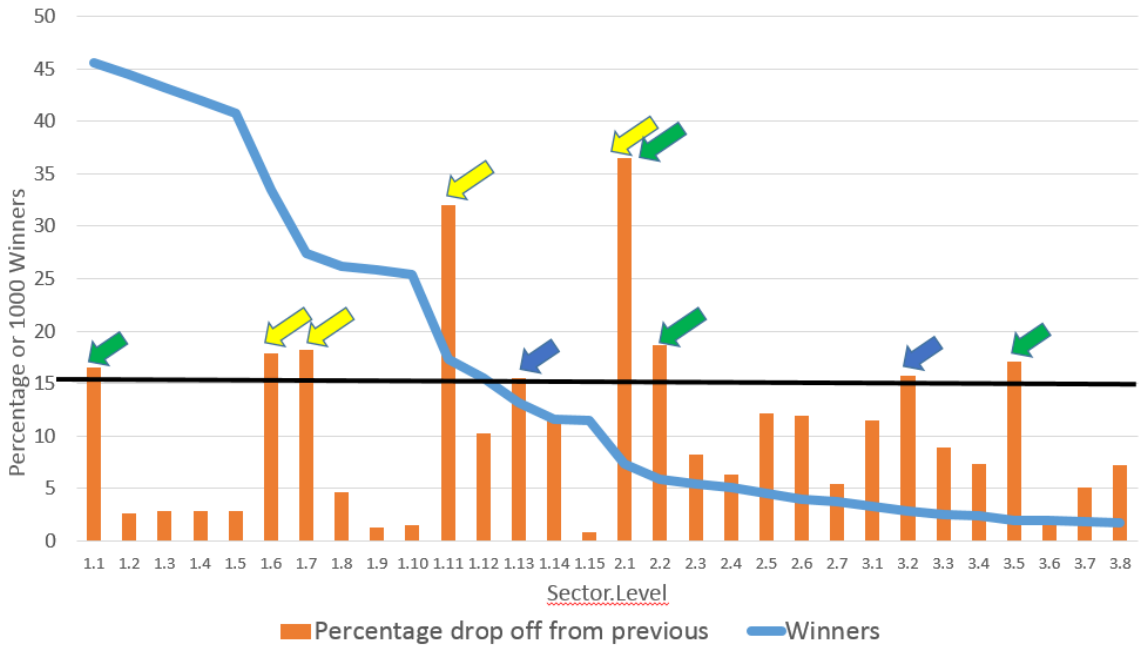

Figure 7 Puzzles with drop off rate higher than 15%

F. User loyalty

We are fortunate to be able to view analytics across a wide range of objects – contests, users, puzzles etc. A figure that we track is how many users do we have per day, and do they return. This information is relevant for the default zone, not for contests, but it verifies the continued interest in Code Hunt from all over the world. We extracted figures for the past month (mid-September to mid-October 2014) when there were an average of 1,150 users per day on the system. Figure 8 shows that on average the ratio of new to returning users is 50%, which is very healthy for a system online.

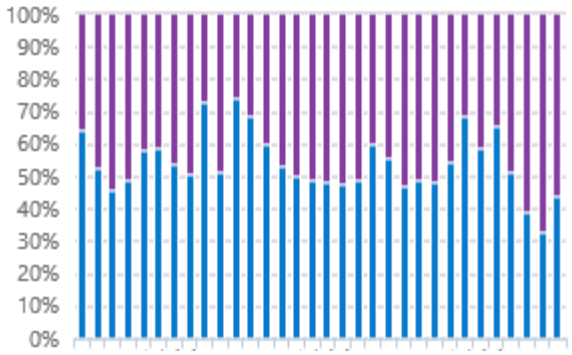

Figure 8 New versus returning users

During this period, the game was used globally, as shown in Figure 9. We have promoted Code Hunt in Europe and North America; the interest in South America grew organically.

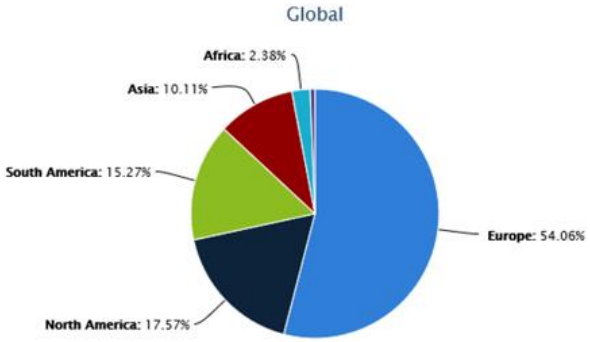

Figure 9 Global usage of Code Hunt

G. Gender Bias

Whenever there is a coding event, questions regarding gender arise. Is the experience attractive to females? Our figures bear out that the number of females playing and enjoying Code Hunt is 12% which is commensurate with the 12.5% reported number of women enrolled in undergraduate degrees in the USA [18]. Code Hunt is globally available, but the USA figures are a good indicator of international trends. Table V shows some figures taken from an open survey with 850 attendees.

TABLE V SURVEY RESULTS BY GENDER

	Female	Male
Respondents	98 or 12%	682 or 80%
Played as practice for a course in C#	11%	18%
Played as practice for a course in Java	40%	23%
Played for own enjoyment	47%	57%

V. *CREATING A CONTEST*

Code Hunt is an open system and it is possible for anyone to create their own contests. Once logged in, there is a window that is accessed through the Settings→Designer buttons. At this stage a window very similar to the main window appears (Figure 3). On the left is described the skeleton of a puzzle, as in Figure 10. First there is the standard blank algorithm that the player sees. The formal parameters and result type of the Puzzle method can be edited to suit different types of puzzles. Then there is the secret solution.

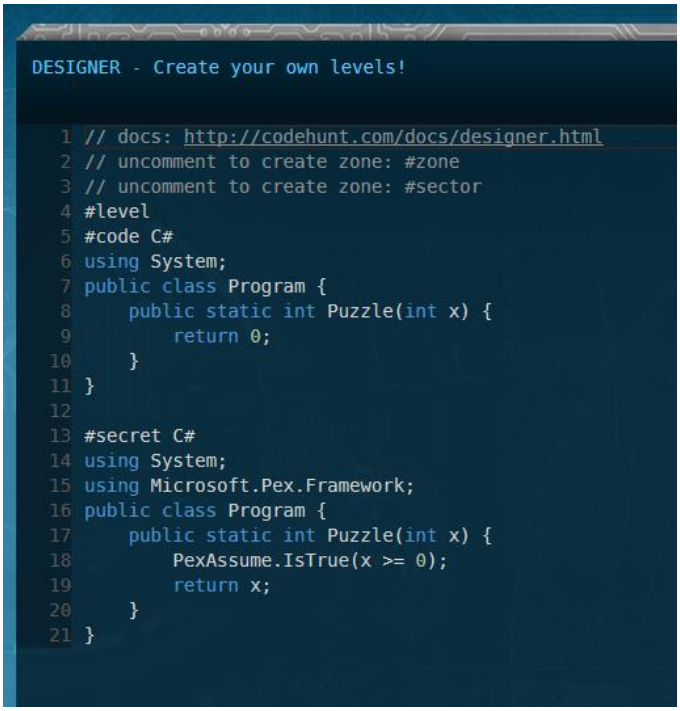

Figure 10 Designing a Puzzle

To prepare a puzzle, the designer creates code in the secret solution and can test it out with the usual Capture Code button, until it is working correctly. Thereafter it can be uploaded. A URL pointing to the puzzle will be returned and it can be shared with others to play.

That experience is for one puzzle. For a complete contest, comprising many puzzles in a zone, they would need to be tested one by one and then the whole list of puzzles, complete with zone and sector tags, would be uploaded. Instructions are given in the URL shown.

While this is the mechanics of creating a contest, finding appropriate puzzles is not easy. Moreover, it is also necessary to make calls on the Pex framework to constrain the trial cases and to add statements which have the effect of suggesting some test cases to Pex. Such test cases will appear in the table of unit test results. The details of this are beyond this paper, but can be found in [13].

VI. *CODE HUNT FOR COURSES*

To assist teaching and learning, our future work is to adopt Code Hunt in courses, in a formal setting (such as college or high school courses) or an informal setting (such as self-training courses for learning specific topics). We have experience of using puzzles in courses through Pex4Fun in a graduate software engineering course with more than 50 enrolled graduate students [17]. The puzzle assignments were organized according to selected topics in software engineering: requirements, design patterns, and testing. The classroom experiences with using puzzles were very encouraging. Grading efforts of the teacher were substantially reduced. More importantly, students' learning process (beyond their final puzzle solutions) along the way was made available to the teacher or teaching assistant. Students had fun while solving problems and improving their learning.

Based on our initial experiences on using puzzles with Pex4Fun in a classroom, we can itemize the features that Code Hunt would need by an instructor who would adopt Code Hunt in a classroom setting. Some of these exist already, but not all.

1) Allow access to student attempts

Allowing access to the sequence of code versions attempted by a student can reveal the puzzle-solving process The instructor can gain insights for diagnosing any learning barriers that the student has and provide customized and personalized guidance to that student, and other students in the class sharing similar problems. Given that manually inspecting the puzzle-solving process can be tedious for a typical-size class, it is desirable to provide tool support to the instructor in order to efficiently discover such insights from the rich data that Code Hunt has. We have already shown in Section IV.C how we can mine the data to make general statements across all players: for a course we need to delve into a particular student's work

2) Integrate instructional material

For a course experience, Code Hunt could provide a seamless integration of course instructional contents and their corresponding puzzles. One way is to have marked up pages with embedded puzzles which can be viewed as interactive textbook pages, providing a learning aspect to Code Hunt as well as a practice aspect. Such pages would alleviate the problem of students encountering new features, as mentioned in Section 0. In a more ambitious effort, in May 2015, Microsoft released the Office Mix add-in for PowerPoint 2013 (https://mix.office.com/). Office Mix allows users (such as instructors) to record audio and/or video of themselves presenting, write on slides as the users speak to them, insert quizzes, polls, online videos, or even Code Hunt puzzles. Figure 11 shows the start of an interactive presentation. Office Mix records every viewing of a Mix, as well as the number of slides seen, and whether the Code Hunt puzzles were solved. These are provided as analytics to the creator of the Mix, as shown in Figure 12. For this Mix, there have been 193 visitors. The column "Answer" provides either the program given by the student or the score, e.g. 127 out of a possible 137.

There is an enhancement coming to Mix where the individual attempts at a puzzle can be seen, as required by 1) above. Interactive textbooks in Code Hunt and interactive presentations in Office Mix are complementary, both enabling effective integration of instructional contents and puzzles.

Figure 11 An Office Mix persentation

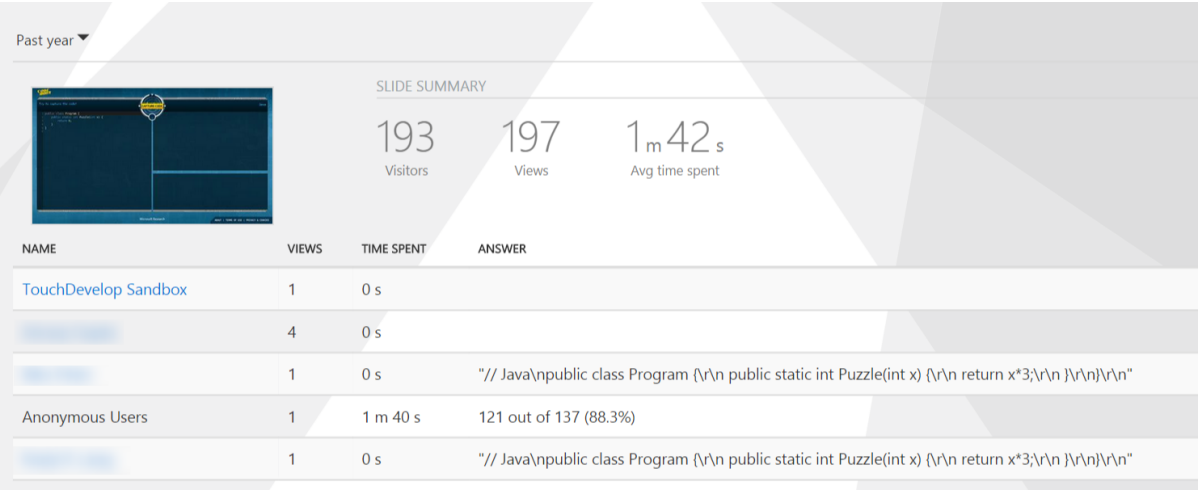

Figure 12 Analytics page from Office Mix

3) Provide hints

In a learning environment it is desirable to provide additional hints to students when they get "stuck" in solving a puzzle. Coding contests are run without such additional hints. However, they are essential in a course setting with focus on improving student learning. Hints can provide the same assistance as an instructor does during office hours when a student drops by to seek help with a coding issue. Code Hunt already has an initial feature for producing additional hints [14]. Our future work plans to further enhance this feature for maximizing student learning effect.

VII. *Related Work*

On-line competitions which exercise the contestants' ability to understand and/or write code have been operating for several years. The typical format of a competition is for a set of algorithmic problems to be described, and then the contestant or a team of contestants have to write and submit code which solves the problems while the clock is ticking. Some of the best known competitions which follow this general pattern include the Data Science track of TopCoder [15], the annual ACM International Collegiate Programming Contest [9], Google Code Jam [6], and the Facebook Hacker Cup [4].

While such competitions serve a valuable purpose in challenging and identifying some of the best programmers in the world, they are not intended to impart any programming skills to the contestants. Many authors claim that competitions like these excite interest in computer programming, but that is about all that can be claimed for them. As Combefis and Wautelet explain [3] these contests do not provide individual feedback to the contestants and do not support training and learning. They applaud Nowicki *et al*. [10], however, for providing an example of a competition style that works. Here, the competitions were held weekly and supplement the teaching in a high school course. The problems match the concepts covered in the course each week. A similar series of competition-style problems related to computer science concepts was described by Voigt *et al.* [16], and these too could be useful as an adjunct to an introductory programming course.

However, none of the competitions mentioned above come close to the approach provided by Code Hunt. The closest in spirit is probably Bug Catcher [2]. In this competition, the contestants do not need any programming skills to enter. They are presented with a series of code fragments which contain bugs, and the goal is to create test cases consisting of inputs and the corresponding expected outputs which illustrate the bugs. The goal of Bug Catcher is to teach software testing topics to high school students in a step by step manner.

Code Hunt is extremely flexible in how it can be used. A graduated series of exercises, which introduce and test ability with a particular programming construct, can be supplied as puzzles to Code Hunt. There is such a set of puzzles which correspond to the Advanced Placement Computer Science course curriculum [1]. Puzzles are grouped into sectors, with each sector exercising a different construct in Java. This is an approach similar to Bug Catcher, but helps students learn programming skills rather than software testing skills. Of course, if the goal is to use Code Hunt to identify fast accurate coders, similarly to Google Code Jam, say, then difficult puzzles which will require contestants to provide non-trivial code as the solutions can be employed. This has been the approach adopted in the Code Hunt competition track of Microsoft's Imagine Cup [7].

VIII. *Conclusions*

Code Hunt is a powerful and versatile platform for coding as a game. It is unique in that requires players to work out what to do from unit tests, rather than giving them a specification. This aspect of Code Hunt adds to the fun, and can be mined in contests. In this paper we described how contests are set up and some of the challenges in ensuring that puzzles are correctly rated. The sheer numbers of players (tens of thousands) make it possible to test hypotheses and come to reasonable conclusions about how players are mastering coding, and what holds them up.

In future work, we are going to perfect the course experience, and also add a club experience, augmenting the do-it-yourself contest process described above. In the back end, we have a system in place that generates hints when users are stuck: it is currently under testing and will be rolled out soon.

At the moment, only C# and Java are supported on Code Hunt. Java programs are actually source translated to C# programs. We have plans to include Python using the same mechanism very soon.

Finally, a responsibility of large games such as these is to keep them fresh. The contests contribute to this, but it is also necessary to periodically update the default zone. A refresh is planned for January 2015. Naturally, care will be taken to preserve the data on sequentially ordering of puzzles so we can continue to delve into it and find out more about how students learn with games.

IX. *Acknowledgments*

We would like to thank our interns Daniel Perelman and Alisha Meherally who were members of the team in the summer of 2014 and made significant contributions to the platform.

X. *References*

[1] AP Computer Science A Course Home Page. http://apcentral.collegeboard.com/apc/public/courses/teachers_corner/4483.html, accessed 21-Oct-2014.

[2] Bryce, R., Andrews, A., Bokser, D., Burton, M., Day, C., Gonzalez, J., Mayo, Q., Noble, T. Bug Catcher: A System for Software Testing Competitions, SIGCSE '13 Proceedings of the 44th ACM technical symposium on Computer Science Education. pp 513-518.

[3] Combéfis, S., Wautelet, J. Programming Trainings and Informatics Teaching Through Online Contests. Proceedings of Olympiads in Informatics, 2014, Vol. 8, pp 21–34.

[4] Facebook Hacker Cup website, https://www.facebook.com/hackercup/, accessed 21-Oct-2014.

[5] Godefroid, P., Klarlund, N., and Sen, K. DART: directed automated random testing. In Proc. PLDI (2005), pp 213–223.

[6] Google Code Jam website, https://code.google.com/codejam, accessed 21-Oct-2014.

[7] Imagine Cup website. https://www.imaginecup.com/, accessed 21-Oct-2014.

[8] Kelleher, C., and Pausch, R. F.: Using storytelling to motivate programming. Comm. ACM, July 2007/Vol. 50, No. 7, pp 58-64.

[9] Official ACM-ICPC website, http://icpc.baylor.edu/, accessed 21-Oct-2014.

[10] Nowicki, M., Matuszak, M., Kwiatkowska, A., Sysło, M., Bała, P. Teaching secondary school students programming using distance learning: a case study. Proceedings of the 10th World Conference on Computers in Education (WCCE 2013).

[11] Tillmann, N., and de Halleux, J. Pex – white box test generation for .NET. In Proc. TAP (2008), 134–153.

[12] Tillmann N, de Halleux J, Xie T, Gulwani S, Bishop J: Teaching and learning programming and software engineering via interactive gaming. ICSE 2013: pp 1117-1126

[13] Tillmann, N., de Halleux, J., Xie, T., and Bishop, J.,, Constructing coding duels in Pex4Fun and code hunt, ISSTA 2014 Proceedings of the 2014 International Symposium on Software Testing and Analysis, pp 445-448

[14] Tillmann, N, de Halleux, J., Bishop, J., Xie, T, Horspool, N., Perelman, D. Code Hunt: Context-Driven Interactive Gaming for Learning Programming and Software Engineering. In Proc. International Workshop on Context in Software Development (CSD 2014), to appearTillmann, N., de Halleux, J., Xie, T., Gulwani, S., and Bishop, J., Teaching and Learning Programming and Software Engineering via Interactive Gaming. In Proc. ICSE (2013), pp 1117–1126.

[15] TopCoder™ website, http://www.topcoder.com, accessed 21-Oct-2104.

[16] Voigt, J., Bell, T., Aspvall, B. Competition-style programming problems for computer science unplugged activities. In: E. Verdu, R. Lorenzo, M. Revilla, L. Regueras (Eds.), A New Learning Paradigm: Competition Supported by Technology. 2010. Boecillo: CEDETEL, 207–234.

[17] Xie, T., Tillmann, N., de Halleux, J., Schulte, W. Fitness-Guided Path Exploration in Dynamic Symbolic Execution. In Proc. DSN 2009, 2009, pp 359-368.

[18] Zweben S, Computing degree and enrollment trends from the 2011-2012 CRA Taulbee Survey